Maker DAO - NST

Security Review

Cantina Managed review by:
Christoph Michel, Lead Security Researcher
M4rio.eth, Security Researcher
Shung, Associate Security Researcher

July 3, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

The Maker Protocol, also known as the Multi-Collateral Dai (MCD) system, allows users to generate Dai (a decentralized, unbiased, collateral-backed cryptocurrency soft-pegged to the US Dollar) by leveraging collateral assets approved by the Maker Governance, which is the community organized and operated process of managing the various aspects of the Maker Protocol.

From May 13th to Jun 14th the Cantina team conducted a review of nst on commit hash 45c9e126.

The Cantina team reviewed MakerDao's NST changes holistically on commit hash b7a5dece9c24fb431352470eb2f04b3ee598ebd8 and determined that all issues were resolved and no new issues were identified.

The team identified a total of **2** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 0
- Gas Optimizations: 1
- Informational: 1

3 Findings

3.1 Gas Optimization

3.1.1 `Nst` version check in deployment script can be simplified

Severity: Gas Optimization

Context: NstInit.sol#L44

Description: The `Nst` deployment script verifies that the deployed `Nst` contract's `version` matches the expected one by checking:

```
require(keccak256(abi.encodePacked(NstLike(instance.nst).version())) == keccak256(abi.encodePacked("1")),
↪  "NstInit/version-does-not-match");
```

Recommendation: The `abi.encodePacked` calls can be replaced with `bytes()` casts:

```
- require(keccak256(abi.encodePacked(NstLike(instance.nst).version())) == keccak256(abi.encodePacked("1")),
↪  "NstInit/version-does-not-match");
+ require(keccak256(bytes(NstLike(instance.nst).version())) == keccak256("1"),
↪  "NstInit/version-does-not-match");
```

Maker: Changed in commit b7a5dece.

Cantina Managed: Fixed.

3.2 Informational

3.2.1 Missing parent initializer call in `Nst`

Severity: Informational

Context: Nst.sol#L66-L69

Description: Upgradeable contracts require replacing the constructor with an initializer function for the implementation contract. The initialize function should call the initialize functions of the inherited parent contracts.

> "Constructors are replaced by internal initializer functions following the naming convention `__{ContractName}_init`. Since these are internal, you must always define your own public initializer function and call the parent initializer of the contract you extend." OZ docs

Recommendation: While the `__UUPSUpgradeable_init` initialization function is a no-op in the used OZ dependency version, consider calling it in `initialize` as a best practice.

Maker: Changed in commit b7a5dece.

Cantina Managed: Fixed.